FACIAL RECOGNITION USING TENSOR-TENSOR DECOMPOSITIONS*

NING HAO†, MISHA E. KILMER‡, KAREN BRAMAN§, AND RANDY C. HOOVER ¶

Abstract. A tensor is a multidimensional array. First order tensors and second order tensors can be viewed as vectors and matrices, respectively. Tensors of higher order, with the ability to include more information, appear more frequently nowadays in image and signal processing, data mining, biomedical engineering and so on. With the recent work of Kilmer and Martin, familiar matrix-based factorizations in linear algebra can be extended in a straightforward way to third order tensors based on their new tensor multiplication and concepts. Our method has an advantage over a popular tensor-based face recognition algorithm called TensorFaces, which is based on the higher-order SVD of an arrangement of the images in the database, in that it does not require a least squares solve for the coefficients. In this paper, we give a brief introduction to the new tensor framework, and apply the induced tensor decompositions to the application of facial recognition. In the numerical results, we compare our new approaches with the traditional PCA method based on matrix decomposition (also known as eigenfaces) and with TensorFaces.

Key words. multilinear algebra, tensor, SVD, PCA, TensorFaces, facial recognition

AMS subject classifications. 15A69, 65F99

1. Introduction. A tensor is a multidimensional array. A first-order tensor is a vector, a second-order tensor is a matrix, and a tensor of order three or higher is a higher-order tensor. In many cases, it is appropriate to store data in higher order multidimensional arrays, rather than as a matricized equivalent form. Consider the example of storing photos of different individuals taken under different lighting conditions with different gestures and viewpoints. The use of a higher order tensor would allow for storage of the images into a fifth order tensor with modes of people, lighting conditions, gestures, viewpoints and pixels [32]. In HITS link analysis [15], use of a third order tensor allows the inclusion of term information to the traditional adjacency matrix of the links between hubs and authorities. In general, higher order tensors arise in a wide variety of application areas, for example in chemometrics [28], psychometrics [17], and image and signal processing [5, 19, 27, 23, 9, 26, 32, 31, 33]. As a result, a number of tensor decompositions such as CANDECOMP/PARAFAC (CP) [3, 8], TUCKER [29], and Higher-Order SVD [18] and tensor-tensor decompositions [13] have been developed to facilitate the extension of linear algebra tools to this multilinear context.

Once higher order tensors have been used to store the inherently multidimensional data, the application will dictate what type of manipulation or post-processing of the data is required. A crucial step in many of the applications involving higher order tensors is multiway compression of the data to ensure the compressed representation of the tensor retains certain characteristics. To this end, many different types of tensor decompositions have been proposed in the literature (see [16] and the references therein).

What is important to note is that the application drives the choice of the decomposition as can be seen in [18, 16]. There is no one-choice-fits-all tensor decomposition. In contrast to the matrix case, where compression is often accomplished via so-called rank-revealing factorizations such as

*Hao and Kilmer's work was supported by NSF grant 0914957

†Department of Mathematics, Tufts University, Medford, MA 02155, ning.hao@tufts.edu,

‡Department of Mathematics, Tufts University, Medford, MA 02155, misha.kilmer@tufts.edu,

§Department of Mathematics and Computer Science, South Dakota School of Mines and Technology, karen.braman@sdsmt.edu,

¶Department of Electrical and Computer Engineering, South Dakota School of Mines and Technology, randy.hoover@sdsmt.edu

the singular value decomposition (SVD), for tensors of order greater than 2, even the concept of rank is a slippery thing (see, for example, the discussion of rank and tensor SVDs in [4] or [16]). CANDECOMP/PARAFAC (CP) decomposition and Tucker decomposition are two of the most widely used tensor decompositions. Given an $I \times J \times K$ tensor $\mathcal{A}$, its CP [16] decomposition is written as a sum of outer products of vectors

$$\mathcal{A} = \sum_{\ell=1}^{R} u^{(\ell)} \circ v^{(\ell)} \circ w^{(\ell)}, \qquad \Longrightarrow \qquad (\mathcal{A})_{i,j,k} = \sum_{\ell=1}^{R} u_i^{(\ell)} v_j^{(\ell)} w_k^{(\ell)},$$

where $u^{(\ell)} \in \mathbb{R}^I, v^{(\ell)} \in \mathbb{R}^J, w^{(\ell)} \in \mathbb{R}^K$ for $\ell = 1, 2, ..., R$. The Tucker decomposition is of the form

$$\mathcal{A} = \mathcal{G} \times_1 \mathbf{U} \times_2 \mathbf{V} \times_3 \mathbf{W} = \sum_{i=1}^{R_1} \sum_{j=1}^{R_2} \sum_{k=1}^{R_3} \sigma_{ijk} (u^{(i)} \circ v^{(j)} \circ w^{(k)})$$

where $R_1 \leq I$, $R_2 \leq J$, $R_3 \leq K$, and for $i, j, k, u^{(i)} \in \mathbb{R}^I, v^{(j)} \in \mathbb{R}^J, w^{(k)} \in \mathbb{R}^K$, $\mathcal{G} = \sigma_{ijk} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$, $\mathbf{U} = [u^{(1)}, u^{(2)}, ..., u^{(R_1)}], \mathbf{V} = [v^{(1)}, v^{(2)}, ..., v^{(R_2)}], \mathbf{W} = [w^{(1)}, w^{(2)}, ..., w^{(R_3)}]$. Both of these decompositions can be seen as higher-order generalization of the matrix singular value decomposition and principal component analysis. However, unlike the matrix SVD whose rank is the number of non-zero singular values, the minimal value of R (the tensor rank of $\mathcal{A}$) in CP cannot always be computed. For the Tucker decomposition, we even need to define a new rank e.g. n-rank to talk about it.

In applications, one often fixes R, and finds the optimal Frobenius norm fit between $\mathcal{A}$ and the sum of rank-1 outer products, with or without additional constraints on the vector components. While this approach may be sufficient for many applications, in this paper we present an entirely different approach based on the framework presented in [13] and obtain energy-revealing factorizations of third-order tensors that we apply to the facial recognition problem. The present work is not the first to consider the use of tensors and tensor factorizations in the context of facial recognition. The first work of this nature of which we are aware took place in the early 2000's [32, 31, 33] and gave rise to an algorithm called TensorFaces. TensorFaces was built around the higher-order SVD [18] and was the first multi-dimensional analog to PCA for facial recognition. Other work in this area includes tensor subspace analysis [24], which is similar to the approach we advocate here in that the images themselves are considered as two-dimensional objects rather than vectorized (they are in traditional matrix-based PCA and in TensorFaces), but differs in that it uses a graph theoretic (Laplacian based) approach.

The remainder of the paper is organized as follows: Section 2 revisits some basic concepts and describes the notation used in the paper. Section 3 describes the traditional matrix-based PCA approach as well as the TensorFaces algorithm, which as mentioned above, was the first multi-dimensional PCA analogue. In Section 4, we review the tensor multiplication and concepts like identity tensor, transpose of a tensor, f-diagonal tensor, and orthogonal tensor necessary to build a tensor SVD. In Section 5 we introduce our new T-SVD based algorithm for facial recognition, and draw comparisons with the matrix-based version. A pivoted tensor QR decomposition is then defined in Section 6 as an alternative tensor decomposition scheme with a potential advantage in updating and downdating the original image data. Performance of all these methods are examined on Extended Yale B dataset and Weizmann dataset in Section 7. Conclusions and future work are the subject of Section 8.

2. Preliminaries. In this section, we cover the basic concepts of tensors and introduce the notation used throughout the paper.

Scalars are denoted by lowercase letters, e.g. a. Vectors are denoted by boldface lowercase letters, e.g. $\mathbf{a}$. Matrices are denoted by boldface capital letters, e.g. $\mathbf{A}$. Higher-order tensors are denoted by Euler script letters, e.g. $\mathcal{X}$. We will use MATLAB conventions for some of the notation and algorithms. For example, the colon notation $\mathbf{A}(:,i)$ means the all rows, column i of $\mathbf{A}$.

The order of a tensor is the dimensionality of the array needed to represent it, also known as ways or modes. An order-p tensor $\mathcal{A}$ is a p-dimensional array. It can be written as

$$\mathcal{A} = (a_{i_1 i_2 \ldots i_p}) \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_p}.$$

In this paper we are primarily interested in decompositions for third order tensors, so the discussion below will focus on blocking third order tensors. We will use MATLAB colon notation to reference the entries in the third-order tensor $\mathcal{A}$.

A fiber is a one-dimensional section of a tensor defined by fixing all indices but one. Third-order tensors have column, row and tube fibers. For instance, the (i,j) tube fiber is denoted $\mathcal{A}(i,j,:)$. Observe that a third order tensor is a matrix of tubal fibers.

A slice of a third order tensor is a two-dimensional section of a third order tensor defined by fixing all indices but two. We use $\mathcal{A}(k,:,:)$, $\mathcal{A}(:,k,:)$ and $\mathcal{A}(:,:,k)$ to denote the horizontal, lateral and frontal slices of a third-order tensor $\mathcal{A}$. $\mathcal{A}(:,:,k)$ is more often denoted compactly as $\mathbf{A}^{(k)}$.

In preparation for defining tensor-tensor multiplication, consider the following. If $\mathcal{A} \in \mathbb{R}^{\ell \times m \times n}$, then

$$\texttt{bcirc}(\mathcal{A}) = \begin{bmatrix} \mathbf{A}^{(1)} & \mathbf{A}^{(n)} & \mathbf{A}^{(n-1)} & \ldots & \mathbf{A}^{(2)} \\ \mathbf{A}^{(2)} & \mathbf{A}^{(1)} & \mathbf{A}^{(n)} & \ldots & \mathbf{A}^{(3)} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \mathbf{A}^{(n)} & \mathbf{A}^{(n-1)} & \mathbf{A}^{(n-2)} & \ldots & \mathbf{A}^{(1)} \end{bmatrix}$$

is a block circulant matrix of size $\ell n \times mn$.

The command $\texttt{unfold}(\mathcal{A})$ takes an $\ell \times m \times n$ tensor into a block $\ell n \times m$ matrix

$$\texttt{unfold}(\mathcal{A}) = \begin{bmatrix} \mathbf{A}^{(1)} \\ \mathbf{A}^{(2)} \\ \vdots \\ \mathbf{A}^{(n)} \end{bmatrix}.$$

The operation $\texttt{fold}$ takes $\texttt{unfold}(\mathcal{A})$ back to tensor form:

$$\texttt{fold}(\texttt{unfold}(\mathcal{A})) = \mathcal{A}.$$

The Frobenious norm of a tensor $\mathcal{A} \in \mathbb{R}^{\ell \times m \times n}$ is

$$||\mathcal{A}||_F = \sqrt{\sum_{i=1}^{\ell} \sum_{j=1}^{m} \sum_{k=1}^{n} a_{ijk}^2},$$

where a_{ijk} is the i,j,k^{th} element of $\mathcal{A}$.

3. Background of PCA-based Approaches. Before we discuss the new tensor decomposition and its relation with principal component analysis, let's begin with a brief explanation of traditional PCA as well as a different, recently-developed higher-order analogue.

3.1. Traditional PCA - Eigenfaces. Principal component analysis was invented in 1901 by Karl Pearson [25], and it has since been a classical technique in image recognition and compression. In simple terms, PCA is used to transform a number of possibly correlated variables into a smaller number of uncorrelated variables known as principal components. The first few principal components explain most of the variation in the original data, while the remaining ones make a negligible contribution. Thus we can describe the data more economically using the first several principal components [30]. We give a brief explanation here in specific context of facial recognition.

Let $\{\mathbf{X}_1, \ldots, \mathbf{X}_m\}$ be a collection of $\ell \times n$ images. Define the matrix $\mathbf{A}$ such that the j^{th} column of $\mathbf{A}$ is $\texttt{vec}(\mathbf{X}_j)$ (i.e. in MATLAB notation, $\mathbf{A}(:, j) := \mathbf{X}_j(:)$). As such, the j^{th} column of $\mathbf{A}$ represents a sample of a multivariate random variable. The sample mean of the set of images is computed as $\mathbf{M} = \frac{1}{m}\sum_{j=1}^{m} \mathbf{A}(:, j)$. If we then update each column of $\mathbf{A}$ according to $\mathbf{A}(:, j) \leftarrow \mathbf{A}(:, j) - \mathbf{M}$, then $\mathbf{A}$ is said to be in mean-deviation form.

The covariance matrix of $\mathbf{A}$ is now given by $\mathbf{C} = \frac{1}{m-1}\mathbf{A}\mathbf{A}^T$. Clearly $\mathbf{C}$ is symmetric positive semidefinite, and is orthogonally diagonalizable: that is, $\mathbf{C} = \mathbf{U}\mathbf{D}\mathbf{U}^T$, where $\mathbf{D} = \texttt{diag}(d_{11}, d_{22}, \cdots)$ is a diagonal matrix containing the eigenvalues of $\mathbf{C}$ with the assumption is that $d_{11} \geq d_{22} \geq \ldots \geq 0$ and $\mathbf{U}$ is an orthogonal matrix. The columns $\mathbf{u}_i$ of the matrix $\mathbf{U}$, which are the eigenvectors of $\mathbf{C}$, are referred to as the principal components, and $\mathbf{u}_1$ denotes the direction in which the data has maximum variance.

Of course for numerical reasons, one does not compute $\mathbf{C}$ and its subsequent eigendecomposition. Rather, given the singular value decomposition $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ of $\mathbf{A}$, it follows from $\mathbf{C} = \frac{1}{m-1}\mathbf{A}\mathbf{A}^T = \frac{1}{m-1}\mathbf{U}\mathbf{S}^2\mathbf{U}^T$ that the non-zero singular values of $\mathbf{A}$ denoted by s_{ii} are related to the non-zero eigenvalues of $\mathbf{C}$ by $s_{ii} = \sqrt{(m-1)d_{ii}}$, and the left singular matrix $\mathbf{U}$ contains the principal components with respect to $\mathbf{C}$.

The basic idea behind the PCA-eigenfaces approach is that $\mathbf{A}(:, j) \approx \mathbf{U}_r\mathbf{U}_r^T\mathbf{A}(:, j)$, where $\mathbf{U}_r := \mathbf{U}(:, 1\!:\!r)$ for some $r \ll m$ because singular values beyond the r^{th} are sufficiently small relative to m, indicating that the vast majority of the variance in the data is well captured in directions corresponding to the largest r singular values. Note that the matrix $\mathbf{U}_r\mathbf{U}_r^T$ is an orthogonal projector onto $\text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$, the space spanned by the first r principal components. Note also that

$$\mathbf{A}(:, j) \approx \mathbf{U}_r(\mathbf{U}_r^T\mathbf{A}(:, j)) = \sum_{i=1}^{r} c_i\mathbf{u}_i, \qquad c_i = \mathbf{u}_i^T\mathbf{A}(:, j) \tag{3.1}$$

which simply states that the (mean subtracted) j^{th} image is well approximated as a linear combination of the orthogonal columns of the $\mathbf{U}_r$ matrix.

If $\mathbf{X}$ denotes a new image (with mean subtracted), then one can compute its expansion coefficients in this space $\mathbf{U}_r^T\texttt{vec}(\mathbf{X})$, and determine the image belongs to the person whose image's expansion coefficient is closest to the coefficient of the testing image.

3.2. TensorFaces. TensorFaces [31] was the first algorithm to handle the facial recognition problem by manipulating tensors. In this algorithm, the data is represented as a tensor with different modes for different factors. One way in which the algorithm is consistent with traditional PCA is that the images are still vectorized. The multidimensionality therefore is only obtained by

Algorithm 1: Traditional Matrix PCA Method

Input: Training: $\mathbf{I}_i$, $i = 1, 2, ..., N$, Testing: $\mathbf{J}$, Truncation index k
Output: Comparison of Test Image with Compressed Training Set

for $i = 1$ **to** N **do**
  $\mathbf{L}(:, i) \leftarrow$ vectorized $\mathbf{I}_i$
end
$\mathbf{M} \leftarrow$ mean image
$\mathbf{A} \leftarrow$ mean-deviation form of $\mathbf{L}$
$\mathbf{U} \leftarrow$ left singular vectors of $\mathbf{A}$
$\mathbf{G} \leftarrow \mathbf{U}(:, 1:k)^T \mathbf{A}$
$\mathbf{T} \leftarrow \mathbf{J} - \mathbf{M}$
$\mathbf{t} \leftarrow$ vectorized form of $\mathbf{T}$
$\mathbf{c} \leftarrow \mathbf{U}(:, 1:k)^T \mathbf{t}$
for $j = 1$ **to** N **do**
  Calculate $||c - \mathbf{G}(:, j)||_F$
end
Claim the training image whose coefficient is closest to that of the test image is the match.

using the other modes to represent other features of the data. The approximation of the tensor is done by use of the multidimensional higher-order SVD (HOSVD), which is a Tucker type of representation (see [16]).

For completeness, the method is given in Algorithm 2 and we describe it briefly now. As is advocated in the original papers, we use a fifth-order tensor $\mathcal{L}$ to specify people, viewpoints, illumination, expression, and pixel respectively, and P, V, I, E are the numbers of people, and number of viewpoints, illumination, expressions for each person in the training set, while P', V', I', E' are those for the testing set. Pix is the length of each vectorized image. Then one calculates the core tensor

$$\mathcal{Z} = \mathcal{L} \times_1 \mathbf{U}_p^T \times_2 \mathbf{U}_v^T \times_3 \mathbf{U}_i^T \times_4 \mathbf{U}_e^T \times_5 \mathbf{U}_{pix}^T \tag{3.2}$$

where $\mathbf{U}_p$, $\mathbf{U}_v$, $\mathbf{U}_i$, $\mathbf{U}_e$, $\mathbf{U}_{pix}$ are the left matrices of the SVD of flattened matrices $\mathbf{L}_{(1)}$, $\mathbf{L}_{(2)}$, $\mathbf{L}_{(3)}$, $\mathbf{L}_{(4)}$, $\mathbf{L}_{(5)}$ of tensor $\mathcal{L}$ along each mode [16, 31]. The core tensor $\mathcal{Z}$ governs the interaction between different factors: $\mathbf{U}_p \in \mathbb{R}^{P \times P}$ spans the space of people parameters, $\mathbf{U}_v \in \mathbb{R}^{V \times V}$ spans the space of viewpoint parameters, $\mathbf{U}_i \in \mathbb{R}^{I \times I}$ spans the space of illumination parameters, $\mathbf{U}_e \in \mathbb{R}^{E \times E}$ spans the space of expression parameters and $\mathbf{U}_{pix} \in \mathbb{R}^{Pix \times (P \times V \times I \times E)}$ spans the space of images. $\mathcal{B} = \mathcal{Z} \times_2 \mathbf{U}_v \times_3 \mathbf{U}_i \times_4 \mathbf{U}_e \times_5 \mathbf{U}_{pix}$ defines $V \times I \times E$ different bases for each combination of viewpoints, illumination and expressions, and sub tensor $\mathcal{B}_{v,i,e}$ is of size $P \times 1 \times 1 \times 1 \times Pix$ for any given v, i, e. For a given image, use flattened matrix $\mathbf{B}_{v,i,e}^{\dagger}$ of $\mathcal{B}_{v,i,e}$ to project it into a set of coefficient vectors $c_{v,i,e}$ for every v, i, e combination. Then the one which yields the smallest $c_{v,i,e} - \mathbf{U}_p(p, :)^T$ identifies the unknown image as person p since row vectors of $\mathbf{U}_p$ are coefficients for each person p.

Unlike the optimal dimensionality reduction in matrix PCA which can be obtained by truncating the singular value decomposition, there is no trivial multilinear counterpart to dimensionality reduction for a Tucker type of factorization. One natural choice for dimensionality reduction is to truncate the mode matrices resulting from the N-mode SVD algorithm, say $\mathbf{U}_p$ is of size $P \times P_1$,

$\mathbf{U}_v$ is of size $V \times V_1$, $\mathbf{U}_i$ is of size $I \times I_1$, $\mathbf{U}_e$ is of size $E \times E_1$, $\mathbf{U}_{pix}$ is of size $Pix \times Pix_1$, where $P_1 \leq P$, $V_1 \leq V$, $I_1 \leq I$, $E_1 \leq E$, $Pix_1 \leq (P \times V \times I \times E)$, then the storage for the decomposition will be reduced to $P_1 \times V_1 \times I_1 \times E_1 \times Pix_1$ (for $\mathcal{Z}$)$+P \times P_1$ (for $\mathbf{U}_p$)$+V \times V_1$ (for $\mathbf{U}_v$)$+I \times I_1$ (for $\mathbf{U}_i$)$+E \times E_1$ (for $\mathbf{U}_e$)$+Pix \times Pix_1$ (for $\mathbf{U}_{pix}$), but the corresponding approximation is generally not optimal in the sense that the corresponding Tucker factorization would be a best approximation to the original in the Frobenius norm. Besides, when it comes to the recognition, we still need to form the basis $\mathcal{B}$ which is of size $P_1 \times V \times I \times E \times Pix$.

Alternatively, one can compute the optimal rank- $(R_1, R_2, R_3, R_4, R_5)$ orthogonal Tucker decomposition, but the optimization is not trivial and requires an iterative algorithm. Furthermore, suitable choices of the truncation values $R_1 - R_5$ are not likely to be known a priori. For details, see [Lathauwer et al. 2000] [20] for an iterative, alternating least squares (ALS) algorithm that improves the mode matrices and hence the core tensor.

Algorithm 2: TensorFaces Method

Input: Training images $\mathbf{I}_{p,v,i,e}$, $p = 1, ..., P$, $v = 1, ..., V$, $i = 1, ..., I$, $e = 1, ..., E$;
Testing Image $\mathbf{J}$
Output: Coefficients of testing images with some tensor basis and comparison

for $p = 1, \ldots, P, v = 1, \ldots, V, i = 1, \ldots, I, e = 1, \ldots, E$ **do**
  $\mathcal{L}(p, v, i, e, :) \leftarrow$ vectorized $\mathbf{I}_{p,v,i,e}$
end
Do the HOSVD to get the core tensor $\mathcal{Z}$ and orthogonal factor matrices $\mathbf{U}_i$:
$\mathcal{L} \leftarrow \mathcal{Z} \times_1 \mathbf{U}_p \times_2 \mathbf{U}_v \times_3 \mathbf{U}_i \times_4 \mathbf{U}_e \times_5 \mathbf{U}_{pix}$
$\mathcal{B} \leftarrow \mathcal{Z} \times_2 \mathbf{U}_v \times_3 \mathbf{U}_i \times_4 \mathbf{U}_e \times_5 \mathbf{U}_{pix}$
$\mathcal{T}(1, 1, 1, 1, :) \leftarrow$ vectorized testing image
for $v = 1, \ldots, V, i = 1, \ldots, I, e = 1, \ldots, E$; **do**
  $c_{v,i,e} \leftarrow (\mathcal{B}^{\dagger}_{v,i,e})^T \mathcal{T}(1, 1, 1, 1, :)$
  for $p = 1, \ldots, P$ **do**
    Calculate $||c_{v,i,e} - \mathbf{U}_p(p, :)^T||_F$
  end
end
Claim the training image whose coefficient is closest to that of the test image is the match.

4. Tensor Framework.

4.1. New Tensor-Tensor Multiplication. We begin this section with a review of the tensor-tensor multiplication [14] and several new concepts based on this concept [13].

DEFINITION 4.1. *Let $\mathcal{A}$ be $\ell \times p \times n$ and $\mathcal{B}$ be $p \times m \times n$, then the tensor-product $\mathcal{A} * \mathcal{B}$ is the $\ell \times m \times n$ tensor*

$$\mathcal{A} * \mathcal{B} = \texttt{fold}(\texttt{bcirc}(\mathcal{A}) \cdot \texttt{unfold}(\mathcal{B})) \tag{4.1}$$

Example: Suppose $\mathcal{A} \in \mathbb{R}^{\ell \times p \times 3}$ and $\mathcal{B} \in \mathbb{R}^{p \times m \times 3}$. Then

$$\mathcal{A} * \mathcal{B} = \texttt{fold}\left(\begin{bmatrix} \mathbf{A}^{(1)} & \mathbf{A}^{(3)} & \mathbf{A}^{(2)} \\ \mathbf{A}^{(2)} & \mathbf{A}^{(1)} & \mathbf{A}^{(3)} \\ \mathbf{A}^{(3)} & \mathbf{A}^{(2)} & \mathbf{A}^{(1)} \end{bmatrix} \begin{bmatrix} \mathbf{B}^{(1)} \\ \mathbf{B}^{(2)} \\ \mathbf{B}^{(3)} \end{bmatrix} \right) \in \mathbb{R}^{\ell \times m \times 3}.$$

With the fact that circulant matrices can be diagonalized with the normalized discrete Fourier transform (DFT) matrix [7], we can block diagonalize block circulant matrices. Suppose $\mathcal{A} \in \mathbb{R}^{\ell \times m \times n}$ and $\mathbf{F}_n$ is the $n \times n$ DFT matrix. Now $\texttt{bcirc}(\mathcal{A})$ is an $n \times n$ block matrix with $\ell \times m$ blocks. Let $\mathbf{P_1}$ and $\mathbf{P_2}$ denote so-called stride permutations such that

$$\mathbf{P_1} \cdot \texttt{bcirc}(\mathcal{A}) \cdot \mathbf{P_2} = \begin{bmatrix} \mathbf{N}_{11} & \mathbf{N}_{12} & \dots & \mathbf{N}_{1m} \\ \mathbf{N}_{21} & \mathbf{N}_{22} & \dots & \mathbf{N}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{N}_{l1} & \mathbf{N}_{l2} & \dots & \mathbf{N}_{lm} \end{bmatrix}$$

is an $\ell \times m$ block matrix with $n \times n$ circulant blocks $\mathbf{N}_{ij}$ where the first column of $\mathbf{N}_{ij}$ is composed of all i, j entries of $\mathbf{A}^{(k)}, k = 1, 2, ..., n$, and

$$\mathbf{P_2}^T \cdot (\mathbf{F}_n^* \otimes \mathbf{I}_m) \cdot \mathbf{P_2} = \mathbf{I}_m \otimes \mathbf{F}_n^*.$$

Observe that

$$\mathbf{P_1} \cdot (\mathbf{F}_n \otimes \mathbf{I}_\ell) \cdot \mathbf{P_1}^T \cdot \mathbf{P_1} \cdot \texttt{bcirc}(\mathcal{A}) \cdot \mathbf{P_2} \cdot \mathbf{P_2}^T \cdot (\mathbf{F}_n^* \otimes \mathbf{I}_m) \cdot \mathbf{P_2}$$

is equal to

$$(\mathbf{I}_\ell \otimes \mathbf{F}_n) \begin{bmatrix} \mathbf{N}_{11} & \mathbf{N}_{12} & \dots & \mathbf{N}_{1m} \\ \mathbf{N}_{21} & \mathbf{N}_{22} & \dots & \mathbf{N}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{N}_{l1} & \mathbf{N}_{l2} & \dots & \mathbf{N}_{lm} \end{bmatrix} (\mathbf{I}_m \otimes \mathbf{F}_n^*) = \begin{bmatrix} \mathbf{D}_{11} & \mathbf{D}_{12} & \dots & \mathbf{D}_{1m} \\ \mathbf{D}_{21} & \mathbf{D}_{22} & \dots & \mathbf{D}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{D}_{l1} & \mathbf{D}_{l2} & \dots & \mathbf{D}_{lm} \end{bmatrix},$$

where each $\mathbf{D}_{ij}$ is an $n \times n$ diagonal matrix. Premultiplying by $\mathbf{P}_1^T$ and postmultiplying by $\mathbf{P}_2^T$,

$$\begin{aligned}(\mathbf{F}_n \otimes \mathbf{I}_\ell) \cdot \texttt{bcirc}(\mathcal{A}) \cdot (\mathbf{F}_n^* \otimes \mathbf{I}_m) &= \mathbf{P_1}^T \begin{bmatrix} \mathbf{D}_{11} & \mathbf{D}_{12} & \dots & \mathbf{D}_{1m} \\ \mathbf{D}_{21} & \mathbf{D}_{22} & \dots & \mathbf{D}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{D}_{l1} & \mathbf{D}_{l2} & \dots & \mathbf{D}_{lm} \end{bmatrix} \mathbf{P_2}^T \\ &= \begin{bmatrix} \hat{\mathbf{A}}^{(1)} & & & \\ & \hat{\mathbf{A}}^{(2)} & & \\ & & \ddots & \\ & & & \hat{\mathbf{A}}^{(n)} \end{bmatrix}.\end{aligned}$$

Applying the fast Fourier transform along the third mode of $\mathcal{A}$ gives us an alternate means of computing the $\hat{\mathbf{A}}^{(i)}$. Let $\hat{\mathcal{A}} = \text{fft}(\mathcal{A}, [\,], 3)$[1], then for each i, $\hat{\mathbf{A}}^{(i)} = \hat{\mathcal{A}}(:, :, i)$. As noted in [13], when the tensors are dense, we could compute $\mathcal{A} * \mathcal{B}$ by computing an FFT along each tubal fiber of $\mathcal{A}$ and $\mathcal{B}$ to obtain $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$, multiply each pair of faces of $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$ to get faces of tensor $\hat{\mathcal{C}}$, then take an inverse FFT along the tubal fiber of $\hat{\mathcal{C}}$ to get the desired result. Since with most applications we have $log(n) \leq m, \ell$, this computation will take $\text{O}(\ell pmn + (\ell p + pm)nlog(n)) \approx \text{O}(\ell pmn)$ compared to $\text{O}(\ell pn^2 m)$ using Definition 4.1. The algorithm for performing the product this way is given in Algorithm 3. It will be useful for understanding the tensor PCA analysis in later sections.

[1]The command $\text{fft}(\mathcal{A}, [\,], 3)$is a MATLAB command that computes the fast Fourier transform (FFT) on the third dimension of a multiway array.

Algorithm 3: Tensor-Tensor Product, using Fourier Domain Computation

Input: $\mathcal{A} \in \mathbb{R}^{\ell \times p \times n}, \mathcal{B} \in \mathbb{R}^{p \times m \times n}$
Output: $\mathcal{C} := \mathcal{A} * \mathcal{B}$
$\hat{\mathcal{A}} \leftarrow \text{fft}(\mathcal{A}, [\,], 3)$; $\hat{\mathcal{B}} \leftarrow \text{fft}(\mathcal{B}, [\,], 3)$
for $i = 1$ **to** n **do**
  $\hat{\mathcal{C}}(:, :, i) \leftarrow \hat{\mathcal{A}}(:, :, i)\hat{\mathcal{B}}(:, :, i)$
end
$\mathcal{C} \leftarrow \text{ifft}(\hat{\mathcal{C}}, [\,], 3)$

Note that due to conjugate symmetry in the Fourier domain, in fact the loop need not run over all n frontal faces. For example, if n is even, we can run the loop for $i = 1, \ldots, (\frac{n}{2}) + 1$, and then populate the remaining $n - (\frac{n}{2} + 1)$ faces as $\hat{\mathcal{C}}(:, :, \frac{n}{2} + j) = \text{conj}(\hat{\mathcal{C}})(:, :, \frac{n}{2} - j - 2)$ for $j = 2, \ldots, n - \frac{n}{2}$.

The next few lemmas and definitions from [14, 13] are necessary for the tensor factorizations we wish to introduce for the purposes of our facial recognition application.

LEMMA 4.2. *$\mathcal{A} * (\mathcal{B} * \mathcal{C}) = (\mathcal{A} * \mathcal{B}) * \mathcal{C}$.*

DEFINITION 4.3. *If $\mathcal{A}$ is $\ell \times m \times n$, then $\mathcal{A}^T$ is the $m \times \ell \times n$ tensor obtained by transposing each of the frontal slices and then reversing the order of transposed frontal slices 2 through n.*

Example: If $\mathcal{A} \in \mathbb{R}^{\ell \times m \times 4}$ and its frontal slices are given by the $\ell \times m$ matrices $\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \mathbf{A}^{(3)}, \mathbf{A}^{(4)}$, then

$$\mathcal{A}^T = \texttt{fold}\left(\begin{bmatrix} (\mathbf{A}^{(1)})^T \\ (\mathbf{A}^{(4)})^T \\ (\mathbf{A}^{(3)})^T \\ (\mathbf{A}^{(2)})^T \end{bmatrix}\right).$$

DEFINITION 4.4. *A tensor is f-diagonal if each frontal slice is a diagonal matrix.*

One special case of an f-diagonal tensor is the identity tensor:

DEFINITION 4.5. *An $\ell \times \ell \times n$ tensor $\mathcal{I}_{\ell\ell n}$ is an identity tensor whose first frontal slice is the $\ell \times \ell$ identity matrix, and whose other frontal slices are all zeros.*

Similarly, a tensor is f-upper triangular if each frontal slice is upper triangular.

Finally, we make significant use in this paper of the concept of an orthogonal tensor:

DEFINITION 4.6. *An $\ell \times \ell \times n$ real valued tensor $\mathcal{Q}$ is orthogonal if*

$$\mathcal{Q}^T * \mathcal{Q} = \mathcal{Q} * \mathcal{Q}^T = \mathcal{I} \tag{4.2}$$

Note that this means the set of all lateral slices (which are themselves matrices) is an orthogonal set, in the sense that

$$\mathcal{Q}(:, i, :)^T * \mathcal{Q}(:, j, :) = \begin{cases} \mathbf{e}_1 & i = j \\ \mathbf{0} & i \neq j \end{cases} \tag{4.3}$$

where $\mathbf{e}_1$ is a tubal tensor with its first frontal slice as 1 and other frontal slices as 0s.

4.2. The T-SVD. In [14, 13], the authors describe a tensor decomposition based on the previously mentioned new tensor-tensor multiplication which is analogous to the matrix SVD. We

now review this decomposition, called the T-SVD. Note that the proof uses the well-known fact that a circulant matrix is diagonalized by the discrete Fourier transform matrix.

THEOREM 4.7 ([13]). *Let $\mathcal{A}$ be an $\ell \times m \times n$ real-valued tensor, then $\mathcal{A}$ can be factored as*

$$\mathcal{A} = \mathcal{U} * \mathcal{S} * \mathcal{V}^T \tag{4.4}$$

with $\ell \times \ell \times n$ orthogonal tensor $\mathcal{U}$, $m \times m \times n$ orthogonal tensor $\mathcal{V}$, and $\ell \times m \times n$ f-diagonal tensor $\mathcal{S}$.

Proof. First, transform $\texttt{bcirc}(\mathcal{A})$ into the Fourier domain:

$$(\mathbf{F}_n \otimes \mathbf{I}_\ell) \cdot \texttt{bcirc}(\mathcal{A}) \cdot (\mathbf{F}_n^* \otimes \mathbf{I}_m) = \begin{bmatrix} \hat{\mathbf{A}}^{(1)} & & & \\ & \hat{\mathbf{A}}^{(2)} & & \\ & & \ddots & \\ & & & \hat{\mathbf{A}}^{(n)} \end{bmatrix}. \tag{4.5}$$

Compute the matrix SVD of each $\hat{\mathbf{A}}^{(i)}$ as $\hat{\mathbf{A}}^{(i)} = \hat{\mathbf{U}}^{(i)}\hat{\mathbf{S}}^{(i)}(\hat{\mathbf{V}}^{(i)})^T$, so

$$\begin{bmatrix} \hat{\mathbf{A}}^{(1)} & & \\ & \ddots & \\ & & \hat{\mathbf{A}}^{(n)} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{U}}^{(1)} & & \\ & \ddots & \\ & & \hat{\mathbf{U}}^{(n)} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{S}}^{(1)} & & \\ & \ddots & \\ & & \hat{\mathbf{S}}^{(n)} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{V}}^{(1)} & & \\ & \ddots & \\ & & \hat{\mathbf{V}}^{(n)} \end{bmatrix}^T, \tag{4.6}$$

where all the matrices in (4.6) are block diagonal.

Now by choosing the proper dimension for $\mathbf{I}$, apply $\mathbf{F}_n^* \otimes \mathbf{I}$ to the left and $\mathbf{F}_n \otimes \mathbf{I}$ to the right of each of the block diagonal matrices to get three block circulant matrices whose first block columns are $\texttt{unfold}(\mathcal{U})$, $\texttt{unfold}(\mathcal{S})$ and $\texttt{unfold}(\mathcal{V}^T)$. Applying `fold` to each of these gives $\mathcal{U}$, $\mathcal{S}$, $\mathcal{V}^T$, respectively.

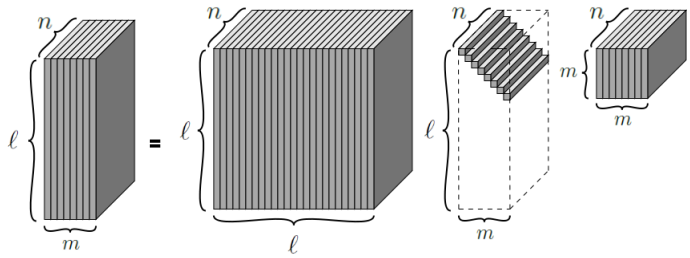

FIG. 4.1. *T-SVD.*

Showing $\mathcal{U}^T * \mathcal{U} = \mathcal{U} * \mathcal{U}^T = \mathcal{I}$, and $\mathcal{V}^T * \mathcal{V} = \mathcal{V} * \mathcal{V}^T = \mathcal{I}$ (i.e. orthogonality of tensors) follows from their definition and the definition of $*$. □

A particularly nice feature of the T-SVD is that it gives a way to find an optimal approximation of a tensor as a sum of $k < \min(n_1, n_2)$ matrix outer products in an analogous way that the matrix SVD can be used to define the optimal approximation of matrix as a sum of vector outer products:

THEOREM 4.8 ([13]). *Let the T-SVD of $\mathcal{A} \in \mathbb{R}^{\ell \times m \times n}$ be given by $\mathcal{A} = \mathcal{U} * \mathcal{S} * \mathcal{V}^T$ and for $k < \min(\ell, m)$ define*

$$\mathcal{A}_k = \sum_{i=1}^{k} \mathcal{U}(:, i, :) * \mathcal{S}(i, i, :) * \mathcal{V}(:, i, :)^T.$$

Then $\mathcal{A}_k = \arg\min_{\tilde{\mathcal{A}} \in M} \|\mathcal{A} - \tilde{\mathcal{A}}\|_F$, *where* $M = \{\mathcal{C} = \mathcal{X} * \mathcal{Y} | \mathcal{X} \in \mathbb{R}^{\ell \times k \times n}, \mathcal{Y} \in \mathbb{R}^{k \times m \times n}\}$. We note that when the 3rd dimension of the tensor is 1 (i.e. $n = 1$), this theorem reduces to the Eckart-Young theorem for matrices, since the t-product in this case becomes the matrix product [13].

5. New T-SVD-based Approach. Our proposed tensor methods differ from matrix-based PCA in that we keep the data in a third-order tensor, and instead use tensor-tensor factorizations based on the $*$ operation. The first approach we propose is based specifically on the T-SVD discussed above. We will draw analogies with the matrix-based PCA approach throughout this section. An alternative (suboptimal) method is given in the next section which is based on a tensor QR factorization. Our T-SVD method also differs from TensorFaces described in the previous section in that a) the images are not pixelated b) we have no extra dimensions for lighting, pose, etc. and c) most significantly, we rely on the T-SVD of the tensor and not the HOSVD of the tensor, as TensorFaces does. What this means is that in our method, we can produce the (tubal) coefficients directly owing to the orthogonality of the "basis" elements (that is, a reconstruction from the compressed information corresponds to an orthogonal projection under the $*$ operation). This means that unlike TensorFaces, we do not have to resort to solving a least squares problem for the coefficients[2], as is done in TensorFaces.

Here we do not vectorize each sample and all decompositions, and the projections are done in tensor space. Because we will be using matrices oriented into the third dimension we will make use of the following notation from [11]: The squeeze operation works on a tensor $\mathcal{X}$ in $\mathbb{R}^{\ell \times 1 \times n}$ to produce a matrix:

$$\mathbf{X} := \texttt{squeeze}(\mathcal{X}) \Rightarrow \mathbf{X}(i,j) := \mathcal{X}(i,1,j)$$

whereas the `twist` operation is the inverse of squeeze

$$\texttt{twist}(\texttt{squeeze}(\mathcal{X})) = \mathcal{X}.$$

Let $\mathbf{M}$ denote the mean image, $\mathcal{A}(:,j,:) = \texttt{twist}(\mathbf{X}_j - \mathbf{M}),\ j = 1, \ldots, m$, then, the samples are still represented by column vectors, but those column vectors are of length ℓ and the entries in the column vector are tube fibers.

In analogy with the covariance matrix in traditional PCA, $\mathcal{K} := \frac{1}{m-1}\mathcal{A} * \mathcal{A}^T$ is symmetric positive semidefinite [11], and is orthogonally diagonalizable [2], i.e. $\mathcal{K} = \mathcal{U} * \mathcal{D} * \mathcal{U}^T$, where $\mathcal{D}$ is an f-diagonal tensor containing the eigen-tuples of $\mathcal{K}$ (i.e. the tube fibers) on the diagonal. The "columns" $\mathcal{U}(:,j,:)$ can be seen as the eigenmatrices of $\mathcal{K}$. This covariance tensor allows us to get the maximum variance directions columnwise because each face $\mathcal{K}(:,:,j)$ measures the covariance of all j-th columns of the images. This idea will be further illustrated in Theorem 5.2 and experiments in the last section.

For reasons similar to the matrix PCA method, we do not compute the eigendecomposition of $\mathcal{K}$ but use the tensor version of the SVD $\mathcal{A} = \mathcal{U} * \mathcal{S} * \mathcal{V}^T$. It follows from $\mathcal{K} = \frac{1}{m-1}\mathcal{A} * \mathcal{A}^T = \frac{1}{m-1}\mathcal{U} * \mathcal{S} * \mathcal{S}^T * \mathcal{U}^T$ that the non-zero eigen-tuples of $\mathcal{K}$ are squares (under the t-product) of the corresponding singular-tuples of $\mathcal{A}$, and the left orthogonal $\mathcal{U}$ contains the principal components with respect to $\mathcal{K}$.

Since $\mathcal{U}$ is orthogonal, $\mathcal{U}(:,1:k,:)$ is an orthogonal set of lateral slices $\mathcal{U}(:,j,:) \in \mathbb{R}^{\ell \times 1 \times n}$, and $\mathcal{U}(:,1:k,:) * \mathcal{U}(:,1:k,:)^T$ defines an orthogonal projector [11] projecting a centered image onto a

[2]The product of $\mathcal{B}^{\dagger}_{i,v,p}$ and current vectorized image represents the solution to a least squares problem.

space with smaller dimension. If we use the first several left singular lateral slices of $\mathcal{U}$ as the new basis, then

$$\mathcal{A}(:,j,:) \approx \mathcal{U}(:,1:k,:) * \mathcal{U}(:,1:k,:)^T * \mathcal{A}(:,j,:) = \sum_{t=1}^{k} \mathcal{U}(:,t,:) * \mathcal{C}(t,j,:) \tag{5.1}$$

where $\mathcal{C}(:,j,:) = \mathcal{U}(:,1:k,:)^T * \mathcal{A}(:,j,:)$ with each tube fiber $\mathcal{C}(t,j,:) = \mathcal{U}(:,t,:)^T * \mathcal{A}(:,j,:)$, $t = 1,2,...,k$. So each centered image $\mathcal{A}(:,j,:)$ can be seen as a t-linear(tensor-linear) combination of orthogonal basis elements $\mathcal{U}(:,t,:)$ for $t = 1,2,\ldots,k$ with the (tubal) coefficients $\mathcal{C}(t,j,:)$. We reduce the recognition problem to comparing coefficients between test images and training images. Note that what must be stored are k, $\ell \times n$ basis elements (used to project incoming images) and a $k \times$ (no. training images) $\times\, n$ tensor $\mathcal{C}$ of coefficients for the training images.

Algorithm 4: T-SVD Method I

Input: Training: $\mathbf{I}_i$, $i = 1,2,...,N$; Test image $\mathbf{J}$, Truncation index k
Output: Match of Test image against Compressed Representation of Training Set

for $i = 1$ **to** N **do**
  $\mathcal{L}(:,i,:) \leftarrow \mathbf{I}_i$
end
$\mathcal{M} \leftarrow$ mean image
$\mathcal{A} \leftarrow$ mean-deviation form of $\mathcal{L}$
$\mathcal{U} \leftarrow$ left singular vectors of tensor $\mathcal{A}$
$\mathcal{C} \leftarrow \mathcal{U}(:,1:k,:)^T * \mathcal{A}$
$\mathcal{T}(:,1,:) \leftarrow \text{twist}(\mathbf{J} - \mathcal{M})$
$\mathcal{B} \leftarrow \mathcal{U}(:,1:k,:)^T * \mathcal{T}$
for $j = 1$ **to** N **do**
  Calculate $||\mathcal{B} - \mathcal{C}(:,j,:)||_F$
end
Claim the training image whose coefficient is closest to that of the test image is the match.

LEMMA 5.1. *When the images are one dimensional signals (i.e. $n = 1$), the tensor SVD method reduces to the traditional PCA method.*

Proof. This follows from the fact that when $n = 1$, the T-SVD reduces to the matrix SVD. □

In our analysis of the approach, it is also useful to re-write (5.1) in matrix-only notation. For convenience, let $\mathbf{Y}_j := \texttt{squeeze}(\mathcal{A}(:,j,:))$. Note that from Algorithm 2, $\mathbf{Y}_j$ is just the jth training image (with mean subtracted). Let $\texttt{circ}(v)$ be the circulant matrix whose first column is v. From (5.1), the definition of $*$ and a little manipulation, it is readily shown that

$$\mathbf{Y}_j \approx \sum_{t=1}^{k} \mathbf{W}_t \texttt{circ}(\mathbf{c}_{t,j}) \tag{5.2}$$

where $\mathbf{W}_t := \texttt{squeeze}(\mathcal{U}(:,t,:))$ is an $\ell \times n$ matrix and ($\mathbf{c}_{t,j}^T := \texttt{squeeze}(\mathcal{C}(t,j,:)^T)$ is a row-vector of length n. We obtain the following result.

THEOREM 5.2. *Let $\mathbf{F}$ denote the $n \times n$ DFT matrix. With $\widehat{\mathbf{Y}_j} := \mathbf{Y}_j\mathbf{F}$, $\widehat{\mathbf{W}_j} := \mathbf{W}_j\mathbf{F}$ and $\widehat{\mathbf{c}_{t,j}}$*

the Fourier coefficients of the length-n column vector $\mathbf{c}_{t,j}$*,*

$$\widehat{\mathbf{Y}_j} \approx \left[\sum_{t=1}^{k} \widehat{\mathbf{c}_{t,j}}^{(1)} \hat{\mathbf{U}}^{(1)}(:,t), \quad \sum_{t=1}^{k} \widehat{\mathbf{c}_{t,j}}^{(2)} \hat{\mathbf{U}}^{(2)}(:,t), \quad \ldots, \quad \sum_{t=1}^{k} \widehat{\mathbf{c}_{t,j}}^{(n)} \hat{\mathbf{U}}^{(n)}(:,t) \right],$$

where $\widehat{\mathbf{c}_{t,j}}^{(i)}$ *refers to the ith component in that vector and* $\widehat{\mathbf{W}_t}(:,i) = \hat{\mathbf{U}}^{(i)}(:,t)$*, where* $\hat{\mathbf{U}}^{(i)}$ *is the unitary matrix from the constructive proof for the T-SVD in equation 4.6. It follows that our Algorithm 2 is equivalent to simultaneous (and independent) PCA on each of the columns of the transformed, mean-subtracted images in Fourier space, using the same truncation index k on each column.*

Proof. Since $\mathbf{Y}_j \approx \sum_{t=1}^{k} \mathbf{W}_t \texttt{circ}(\mathbf{c}_{t,j})$, we have

$$\mathbf{Y}_j\mathbf{F} \approx \sum_{t=1}^{k} \mathbf{W}_t \texttt{circ}(\mathbf{c}_{t,j})\mathbf{F} = \sum_{t=1}^{k} \mathbf{W}_t \mathbf{F}\mathbf{F}^H \texttt{circ}((\mathbf{c}_{t,j})\mathbf{F} = \sum_{t=1}^{k} \widehat{\mathbf{W}_t} \texttt{diag}(\widehat{\mathbf{c}_{t,j}})$$

i.e.

$$\begin{aligned} \widehat{\mathbf{Y}_j} &\approx \sum_{t=1}^{k} [\widehat{\mathbf{c}_{t,j}}^{(1)} \widehat{\mathbf{W}_t}(:,1), \widehat{\mathbf{c}_{t,j}}^{(2)} \widehat{\mathbf{W}_t}(:,2), \ldots, \widehat{\mathbf{c}_{t,j}}^{(n)} \widehat{\mathbf{W}_t}(:,n)], \qquad (5.3) \\ &= \left[\sum_{t=1}^{k} \widehat{\mathbf{c}_{t,j}}^{(1)} \hat{\mathbf{U}}^{(1)}(:,t), \quad \sum_{t=1}^{k} \widehat{\mathbf{c}_{t,j}}^{(2)} \hat{\mathbf{U}}^{(2)}(:,t), \quad \ldots, \quad \sum_{t=1}^{k} \widehat{\mathbf{c}_{t,j}}^{(n)} \hat{\mathbf{U}}^{(n)}(:,t) \right], \end{aligned}$$

The rest of the claim follows directly. □

A major consequence of this theorem is that it illustrates the potential for further compression of the information in the database, provided we are willing to work in Fourier space directly. To see this, first observe that there is no loss in the recognition procedure if we operate in the Fourier domain (we will assume n is even for simplicity):

- The comparison of a test image $\mathbf{J}$'s tubal expansion coefficients to the training coefficient tensor can be made in the Fourier domain. Observe that the product $\mathcal{Z} := \mathcal{U}(:,1:k,:)^T * \text{twist}(\mathbf{J} - \mathbf{M})$ can be computed using Algorithm 3, where we accumulate the $k \times 1 \times n$ object $\hat{\mathcal{Z}}$ in the Fourier domain anyway.
- As noted in the discussion following Algorithm 3, due to conjugate symmetry, only about half of the matrix-vector products (i.e. the products inside the loop) need to be executed to compute $\hat{\mathcal{Z}}$. This means only the first $\frac{n}{2}+1$ frontal slices of $\hat{\mathcal{U}}(:,1:k,:)$ need to be saved.
- If we want to compare $\mathcal{Z}$ with the lateral slices of $\mathcal{C}$ to make comparisons, we can equivalently compare $\hat{\mathcal{Z}}(:,1,1:\frac{n}{2}+1)$ to $\hat{\mathcal{C}}(:,j,1:\frac{n}{2}+1)$ for $j = 1, \ldots,$ no. training images.

Storing the first $\frac{n}{2}+1$ frontal slices of $\hat{\mathcal{U}}(:,1:k,:)$ and $\hat{\mathcal{C}}$ requires the same amount of storage on the computer as the storage of $\mathcal{U}$ and $\mathcal{C}$. This is because, except for the DC term and $\frac{n}{2}+1$ term, we must store complex double precision numbers. But, since the theorem says we are doing independent PCA on each of the columns of the transformed, mean subtracted images in Fourier space, we decide it makes more sense to vary k across the columns. Specifically, we would replace (5.3) with

$$\widehat{\mathbf{Y}_j} \approx [\sum_{t=1}^{k_1} \widehat{\mathbf{c}_{t,j}}^{(1)} \hat{\mathbf{U}}^{(1)}(:,t), \sum_{t=1}^{k_2} \widehat{\mathbf{c}_{t,j}}^{(2)} \hat{\mathbf{U}}^{(2)}(:,t), \sum_{t=1}^{k_3} \widehat{\mathbf{c}_{t,j}}^{(3)} \hat{\mathbf{U}}^{(3)}(:,t), \ldots, , \sum_{t=1}^{k_n} \widehat{\mathbf{c}_{t,j}}^{(n)} \hat{\mathbf{U}}^{(n)}(:,t)], \quad (5.4)$$

where we are sure that, to preserve conjugate symmetry, the truncation index for column j, $2 \leq j \leq \frac{n}{2}$ is the same as for column $n - j + 2$.

In fact, because of conjugate symmetry and the need to compute/match against only roughly the first half of the coefficients for each image, we need not explicitly form/store the last $n - \frac{n}{2} - 1$ columns. Thus, the storage requirements are broken down as follows:

- For a given $\ell \times n$ image in the training set, and $\frac{n}{2}+1$ truncation indices $k_1, \ldots, k_{\frac{n}{2}+1}$, with $k_1 \geq k_i$, we need to store

$$K = k_1 + 2k_2 + \ldots + 2k_{\frac{n}{2}} + k_{\frac{n}{2}+1} \tag{5.5}$$

  real double precision numbers (a complex store is equal to 2 reals, hence the factor of 2) to represent the non-truncated coefficients in Fourier space.
- From each $\hat{\mathbf{U}}^{(i)}$, $i = 1, \ldots, \frac{n}{2}+1$, we need save only k_i columns. This requires storing ℓK double precision numbers.
- So for $m =$ no. training images, we must store $\ell K + mK$ double precision numbers in total.

It remains to consider how the truncation parameters k_i, $i = 1, \ldots, \frac{n}{2}+1$, might be identified. Recall that $\hat{\mathcal{A}}$ is $\ell \times m \times n$. Note that from Theorem 4.7, we have that $\hat{\mathbf{U}}^{(i)}$ contains the left singular vectors of the ith frontal slice of $\hat{\mathcal{A}}$, and $\hat{\mathbf{S}}^{(i)}$ contains the corresponding singular values $\hat{\mathbf{s}}_j^{(i)}$, $j = 1, \ldots, \min\{\ell, m\}$ for the ith frontal slice. Next, observe that

$$\|\hat{\mathcal{A}}(:,:,1:\frac{n}{2}+1)\|_F^2 = \sum_{i=1}^{\frac{n}{2}+1} \|\hat{\mathcal{A}}(:,:,i)\|_F^2 = \sum_{i=1}^{\frac{n}{2}+1} \sum_{j=1}^{\min(\ell,m)} (\hat{\mathbf{s}}_j^{(i)})^2. \tag{5.6}$$

Hence, in order to optimally define the truncation indices, consider the relative energy measure

$$\frac{\sum_{i=1}^{\frac{n}{2}+1} \sum_{j=1}^{k_i} (\hat{\mathbf{s}}_j^{(i)})^2}{\|\hat{\mathcal{A}}(:,:,1:\frac{n}{2}+1)\|_F^2}. \tag{5.7}$$

We should clearly keep the largest (considered over *both* indices i and j) singular values in order to maintain a relative energy close to 1. Therefore, we decide on a desired relative energy value (say .9), order the $\hat{\mathbf{s}}_j^{(i)}$ in a vector $\mathbf{q}$ from largest to smallest, find index t to be the smallest index such that $\sum_{i=1}^{\frac{n}{2}+1} \sum_{j=1}^{k_i} \{(\hat{\mathbf{s}}_j^{(i)})^2 : (\hat{\mathbf{s}}_j^{(i)})^2 > \mathbf{q}(t)^2\} > \|\hat{\mathcal{A}}(:,:,1:\frac{n}{2}+1)\|_F^2 \times .9$. Recall that for each fixed i, $\hat{\mathbf{s}}_1^{(i)} \geq \hat{\mathbf{s}}_2^{(i)} \geq \ldots \hat{\mathbf{s}}_{\min\{\ell,m\}}^{(i)}$. Thus, the k_i are defined so that $\hat{\mathbf{s}}_{k_i}^{(i)} > \mathbf{q}(t)$ but $\hat{\mathbf{s}}_{k_{i+1}}^{(i)} \leq \mathbf{q}(t)$. The new method based on this strategy is given in Algorithm 5.

As a final note, although in Algorithm 5 we are advocating working in the Fourier domain, we can make a comparison with something happening in the spatial domain. Define a f-diagonal tensor $\mathcal{D}$ so that the Fourier coefficients along each diagonal tube fiber satisfy, for $i = 1, \ldots, \frac{n}{2}+1$

$$\hat{\mathbf{d}}_j^{(i)} = \begin{cases} 1 & \text{if } \mathbf{s}_j^{(i)} \text{ is kept} \\ 0 & \text{if } \mathbf{s}_j^{(i)} \text{ is discarded through procedure above.} \end{cases},$$

and set $\hat{\mathbf{d}}_j^{(n-i+2)} = \hat{\mathbf{d}}_j^{(i)}$ to ensure conjugate symmetry. Then in the spatial domain, the compressed test set is (implicitly) represented using orthogonal projection of $\mathcal{A}$ onto a particular range-space, i.e. in the spatial domain, the representation can be written $(\mathcal{U}(:,1:k,:) * \mathcal{D} * \mathcal{U}(:,1:k,:)^T) * \mathcal{A}$ where the tensor in parenthesis is an orthogonal projector (see [12] for more details).

Algorithm 5: T-SVD Method II

Input: Training: $\mathbf{I}_i$, $i = 1, 2, ..., N$; Test image $\mathbf{J}$, Truncation index k

Output: Fourier Domain Comparison of Test image against Compressed Representation of Training Set

for $i = 1$ **to** N **do**
  $\mathcal{L}(:, i, :) \leftarrow \mathbf{I}_i$
end
$\mathcal{M} \leftarrow$ mean image
$\mathcal{A} \leftarrow$ mean-deviation form of $\mathcal{L}$
Frontal face SVDs of $\hat{\mathbf{A}}^{(i)} \leftarrow \hat{\mathcal{A}}(:, :, i)$, $i = 1, \ldots \frac{n}{2} + 1$
Employ drop strategy; keep only $\hat{\mathbf{U}}^{(i)}(:, 1 : k_i)$
$\mathcal{T}(:, 1, :) \leftarrow \text{twist}(\mathbf{J} - \mathcal{M})$
for $i = 1 : \frac{n}{2} + 1$, $t = 1 : k_i$, $j = 1 : N$ **do**
  $\hat{\mathbf{c}}_{t,j}^{(i)} \leftarrow (\hat{\mathbf{U}}^{(i)}(:, t))^H \hat{\mathbf{A}}^{(i)}(:, j)$
end
for $i = 1 : \frac{n}{2} + 1$, $t = 1 : k_i$, **do**
  $\hat{\mathbf{b}_t}^{(i)} \leftarrow (\hat{\mathbf{U}}^{(i)}(:, t))^H \hat{\mathbf{T}}^{(i)}(:, 1)$
end
for $j = 1 : N$ **do**
  Calculate $||\hat{\mathbf{b}_t}^{(i)} - \hat{\mathbf{c}}_{t,j}^{(i)}||_F$
end
Claim the training image whose coefficient is closest to that of the test image is the match.

6. Tensor Pivoted QR Alternative. In this section we propose one possible alternative, a T-PQR (tensor (pivoted) QR) factorization, to the T-SVD for our application. The reason for proposing a T-PQR alternative is the potential advantage in updating or downdating the face database. The disadvantage is that this factorization will not be optimal. Pictorially, adding or deleting a new lateral slice $\mathcal{L}$ to the database $\mathcal{A}$ of size $\ell \times m \times n$, is illustrated in Fig. 6.1.

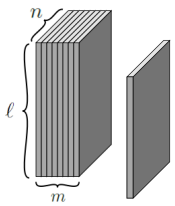

FIG. 6.1. *Updating and Downdating*

Before we begin, we wish to define a permutation tensor. A permutation tensor $\mathcal{P} \in \mathbb{R}^{m \times m \times n}$ is a tensor whose entries consist of only zeros and 1's, and for which $\mathcal{P}^T * \mathcal{P} = \mathcal{P} * \mathcal{P}^T = \mathcal{I}$.

THEOREM 6.1. *Let $\mathcal{A}$ be an $\ell \times m \times n$ real-valued tensor, then $\mathcal{A}$ can be factored as*

$$\mathcal{A} * \mathcal{P} = \mathcal{Q} * \mathcal{R} \tag{6.1}$$

where $\mathcal{Q}$ is an $\ell \times \ell \times n$ orthogonal tensor, $\mathcal{R}$ is an $\ell \times m \times n$ f-upper triangular tensor, and $\mathcal{P}$ is a permutation tensor.

Proof. The proof when $\mathcal{P} = \mathcal{I}$ is included in [14]. For the more general case, as in the T-SVD proof, we move to the Fourier domain as in (4.5). Then we can get the column pivoted QR-factorization $\hat{\mathbf{A}}^{(1)}\mathbf{P}^{(1)} = \hat{\mathbf{Q}}^{(1)}\hat{\mathbf{R}}^{(1)}$. We first apply the same permutation to every other frontal slice of $\hat{\mathcal{A}}$, then compute the QR factorization of each permuted face

$$\hat{\mathbf{A}}^{(i)}\mathbf{P}^{(1)} = \hat{\mathbf{Q}}^{(i)}\hat{\mathbf{R}}^{(i)}.$$

Let $\mathbf{P}$, $\hat{\mathbf{Q}}$ and $\hat{\mathbf{R}}$ be the block diagonal matrices with $\mathbf{P}^{(1)}$, $\hat{\mathbf{Q}}^{(i)}$ and $\hat{\mathbf{R}}^{(i)}$ down the main block diagonals, and apply $\mathbf{F}^* \otimes \mathbf{I}$ to the left and $\mathbf{F} \otimes \mathbf{I}$ to the right of each of these block diagonal matrices to get block circulant matrices, respectively. Then we get $\mathcal{P}, Q, R$ by folding the first block-column of the block circulants respectively. Note that, since $\mathbf{P}$ was constant along the block diagonal, $\mathcal{P}$ must be a permutation tensor, as desired. Furthermore, $\mathcal{P}$ will have $\mathbf{P}^{(1)}$ in the front face and zeros elsewhere, so that $\mathcal{A} * \mathcal{P}$ is equivalent to permuting the lateral slices. □

The potential use of a permutation tensor is to permute lateral slices such that more of the energy is contained in the first few lateral slices of the orthogonal $\mathcal{Q}$ tensor. This is similar to using column permutation with matrix-QR algorithms to achieve better rank-revealing properties. In analogy with Algorithm 4, we arrive at Algorithm 6. We can also derive a further compressed representation along the lines of Algorithm 5, where we operate in the Fourier domain, exploit conjugate symmetry, and save only $\hat{\mathbf{Q}}^{(i)}(:, 1 : k_i)$, where the k_i are arrived at using the same drop strategy as before but on the diagonal elements of all the $\hat{\mathbf{R}}^{(i)}$. In the interest of space, we omit that variant of the T-PQR algorithm.

Algorithm 6: Tensor QR Method

Input: Training: $\mathbf{I}_i$, $i = 1, 2, ..., N$, Testing: $\mathbf{J}$, Truncation index k
Output: Coefficients of testing images with a basis of reduced dimension

for $i = 1$ **to** N **do**
  $\mathcal{L}(:, i, :) \leftarrow \mathbf{I}_i$
end
$\mathcal{M} \leftarrow$ mean image
$\mathcal{A} \leftarrow$ mean-deviation form of $\mathcal{L}$
$[\mathcal{Q}, \mathcal{R}] \leftarrow$ tensor (pivoted) QR decomposition of $\mathcal{A}$
$\mathcal{C} \leftarrow \mathcal{Q}(:, 1 : k, :)^T * \mathcal{A}$
$\mathcal{T}(:, 1, :) \leftarrow \text{twist}(\mathbf{J} - \mathcal{M})$
$\mathcal{B} \leftarrow \mathcal{Q}(:, 1 : k, :)^T * \mathcal{T}$
for $j = 1$ **to** N **do**
  Calculate $||\mathcal{B} - \mathcal{C}(:, j, :)||_F$
end
Claim the training image whose coefficient is closest to that of the test image is the match.

In terms of a discussion of updating, we will assume that the most common case in practice will be the need to update the database, which increases the lateral dimension by 1, so that an $\ell \times m \times n$ database becomes an $\ell \times (m + 1) \times n$ database. Adding an image means (implicitly) we have one more lateral slice to the original tensor. As the factorization (and therefore updates to the previous factorization) are done in the Fourier domain, we first must take an extra FFT along tube

fibers of the new image slice to move it into the Fourier domain. Once in the Fourier domain, a new column is appended to each $\hat{\mathbf{R}}^{(i)}$, and the standard matrix pivoted QR update is performed to update $\mathbf{P}^{(1)}, \hat{\mathbf{Q}}^{(1)}, \hat{\mathbf{R}}^{(1)}$. The updated permutation is next applied to the recently augmented $\hat{\mathbf{R}}^{(i)}$ and updates to the $\hat{\mathbf{Q}}^{(i)}$ and $\hat{\mathbf{R}}^{(i)}$ are also computed. In other words, the benefit of using matrix pivoted QR on the matrix $\hat{\mathbf{A}}^{(i)}$ in the Fourier domain vs. the SVD on each of these matrices is that it is much cheaper to update the pivoted QR factorization in general than the SVD.

7. Experiments. In this section, we apply the algorithms mentioned above on the Extended Yale B database [1, 6, 21] and Weizmann database.

As explained in Section 4, there is no trivial dimensionality reduction method for the HOSVD used by the TensorFaces algorithm. Although we could truncate the mode matrices or try the optimal reduced rank approximation, when it comes to the facial recognition application, we still need to form the basis $\mathcal{B}$ which by definition is of size $J_1 \times I_2 \times \cdots \times I_N$, where J_1 is the reduced first dimension, and I_i's are the original dimensions. So most of its advantages lie in data compression. Based on these considerations, we will compare the storage and recognition rates of our truncated tensor-based algorithm against traditional PCA in the first example, and we will compare TensorFaces with our new tensor-based algorithm in the second and third examples.

At first glance, a potential downside of our T-SVD based and T-PQR based algorithms is that it is tagged to the orientation of the images in the tensor database. Indeed, if we rotated the images the same way prior to placing them in the database, we would expect the results (i.e. tensor basis, coefficient tensors) to be different. We will show that the performance of our tensor algorithms does not deteriorate if we, say, transpose the images prior to putting putting them in the tensor. This makes sense, because it is equivalent to doing Fourier transforms along columns of an image vs. rows prior to the computations/factorizations of frontal slices of the tensor. Given the nature of these images, we would expect the information content in the Fourier domain under rotation of the images not to be very different. On the other hand, one would not want to construct the database by arranging it so that the images become the frontal slices of the tensor, because then we lose the ability to describe the images as (near) t-linear combinations of the orthogonal basis-like columns of $\mathcal{U}$ as in (5.1). In all our experiments, we measure the recognition ratio by

$$\frac{\text{number of correctly matched images}}{\text{number of test images}}.$$

As we have mentioned before, we match a testing image to the person whose image's coefficient is closest, in the Frobenius norm, to the coefficient of the testing image. The matches are counted on an individual image basis, and the denominator is with respect to the total number of test images in each trial.

7.1. Facial Recognition on the Extended Yale B Database. We experiment on a subset of the Extended Yale B Database which contains the first 20 different illuminations of all 38 people, and we decimate the images by a factor of 3. Sample images from the data set are illustrated in Fig. 7.1.

In Experiment 1, for each trial, we randomly select 15 illumination conditions, and use the corresponding images for each person as the training set. The images corresponding to the remaining 5 illumination conditions for each person are used for the test. In Experiment 2, for each trial, we randomly select 5 illumination conditions, and use the corresponding images of each person as the training set. The images corresponding to the remaining 15 illumination conditions for each person are used for the test. In both experiments, we run 20 trials and we use the relative energy measure

given by (5.7) for a desired .9 threshold to decide $k_i's$ and K (see equations (5.7) and (5.5)) for T-SVD and T-PQR.

FIG. 7.1. *Sample images from the Extended Yale B dataset*

The averaged results of T-SVD vs. PCA methods, and T-PQR vs. PCA methods over the 20 trials are respectively listed in Table 7.1, 7.2 and Table 7.5, 7.6. For the same recognition rate, we list the number K from (5.5) in our tensor-based methods and the number k used in PCA, as well as their corresponding storage figures. Here the storage of double precision numbers is calculated as follows:

$$\texttt{Storage}_{\texttt{T-SVD}} = \texttt{K}_{\texttt{T-SVD}} \times \ell + \texttt{K}_{\texttt{T-SVD}} \times (\texttt{number of training images}),$$

$$\texttt{Storage}_{\texttt{T-PQR}} = \texttt{K}_{\texttt{T-PQR}} \times \ell + \texttt{K}_{\texttt{T-PQR}} \times (\texttt{number of training images}),$$

$$\texttt{Storage}_{\texttt{PCA}} = \texttt{k}_{\texttt{PCA}} \times (\ell \times \texttt{n}) + \texttt{k}_{\texttt{PCA}} \times (\texttt{number of training images}).$$

The details of 10 out of the 20 random results are listed in Table 7.3, 7.4 and Table 7.7, 7.8. For the same amount of storage, the recognition rates based on tensor methods and the PCA method are shown in Fig. 7.2 and Fig. 7.3.

	Rec Rate	K by T-SVD	Storage for T-SVD	k by PCA	Storage for PCA
mean	0.79	103	65334	36	148921
median	0.817	103	65302	35	145390
max	0.932	114	72276	50	207700
min	0.511	91	57694	30	124620

TABLE 7.1
Comparison between T-SVD (Algorithm 5) and PCA (Algorithm 1) in Experiment 1.

As mentioned at the beginning of this section, it is important to place the images into the 3rd order tensor as lateral slices. Given this restriction, however, one reasonable question is whether or not the orientation of the images (transposed or upright) makes a difference with our approach. Clearly, if the images are not square, the tensor dimensions change (an $m \times \ell \times n$ tensor becomes $n \times \ell \times m$). This means the $\mathcal{U}$ and $\mathcal{S}$ would be different, since those tensors are determined by first applying FFTs along tube fibers of the tensor: in the former case, taking FFTs along tube fibers corresponds to taking FFTs along rows of the images which have been placed as lateral slices into

	Rec Rate	K by T-SVD	Storage for T-SVD	k by PCA	Storage for PCA
mean	0.656	110	27965	36	135864
median	0.675	113	28702	36	135864
max	0.760	138	35052	44	166056
min	0.558	81	20574	29	109446

TABLE 7.2
Comparison between T-SVD (Algorithm 5) and PCA (Algorithm 1) in Experiment 2.

Rec Rate	0.842	0.737	0.816	0.868	0.69	0.742	0.932	0.511	0.858	0.837
K, T-SVD	91	106	95	112	114	101	108	101	104	100
Storage	57694	67204	60230	71008	72276	64034	68472	64034	65936	63400
k, PCA	30	31	30	44	42	35	40	35	39	39
Storage	124620	128774	124620	182776	174468	145390	166160	145390	162006	162006

TABLE 7.3
10 out of 20 random results of T-SVD (Algorithm 5) and PCA (Algorithm 1) in Experiment 1.

Rec Rate	0.7	0.695	0.642	0.709	0.76	0.577	0.695	0.688	0.593	0.695
K, T-SVD	123	116	97	95	115	81	116	118	138	118
Storage	31242	29464	24638	24130	29210	20574	29464	29972	35052	29972
k, PCA	42	41	33	35	40	29	41	40	36	36
Storage	158508	154734	124542	132090	150960	109446	154734	150960	135864	135864

TABLE 7.4
10 out of 20 random results of T-SVD (Algorithm 5) and PCA (Algorithm 1) in Experiment 2.

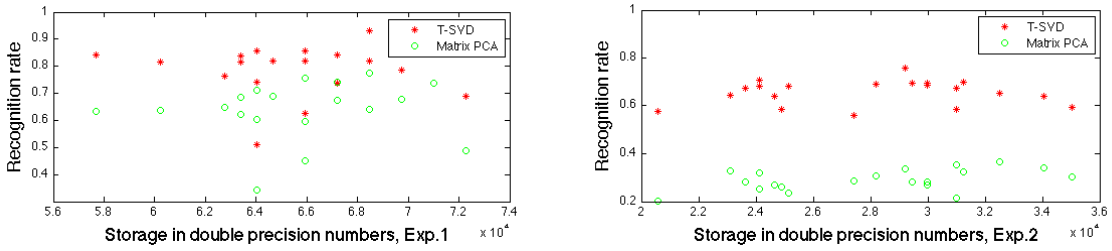

FIG. 7.2. *Recognition Rate and Storage (T-SVD vs PCA), Experiment 1 (left) and Experiment 2 (right)*

	Rec Rate	K by T-PQR	Storage for T-PQR	k by PCA	Storage for PCA
mean	0.825	216	136786	49	201261
median	0.847	216	136627	47	195238
max	0.947	241	152794	71	294934
min	0.579	193	122362	35	145390

TABLE 7.5
Comparison between T-PQR (Algorithm 6 with compression) and PCA (Algorithm 1) in Experiment 1.

the tensor, and in the latter case, it is equivalent to taking FFTs along the rows of the transposed images (i.e. along the columns of the images). Thus, our algorithm is not invariant to transposition. On the other hand, in the average situation where the number of rows and columns of the images

	Rec Rate	K by T-PQR	Storage for T-PQR	k by PCA	Storage for PCA
mean	0.709	261	66294	49	183794
median	0.717	275	69723	47	177378
max	0.814	304	77216	59	222666
min	0.597	197	50038	39	147186

TABLE 7.6

Comparison between T-PQR (Algorithm 6 with compression) and PCA (Algorithm 1) in Experiment 2.

Rec Rate	0.895	0.784	0.579	0.847	0.874	0.811	0.947	0.89	0.853	0.884
K:T-PQR	214	209	230	203	202	208	216	207	224	219
Storage	135676	132506	145820	128702	128068	131872	136944	131238	142016	138846
k:PCA	41	48	59	35	44	44	45	54	46	44
Storage	170314	199392	245086	145390	182776	182776	186930	224316	191084	182776

TABLE 7.7

10 out of 20 random results of T-PQR (Algorithm 6 with compression) and PCA (Algorithm 1) in Experiment 1.

Rec Rate	0.733	0.677	0.749	0.814	0.761	0.714	0.679	0.597	0.779	0.754
K, T-PQR	244	216	291	284	287	289	273	231	235	240
Storage	61976	54864	73914	72136	72898	73406	69342	58674	59690	60960
k, PCA	46	41	57	58	59	46	57	47	48	51
Storage	173604	154734	215118	218892	222666	173604	215118	177378	181152	192474

TABLE 7.8

10 out of 20 random results of T-PQR (Algorithm 6 with compression) and PCA (Algorithm 1) in Experiment 2.

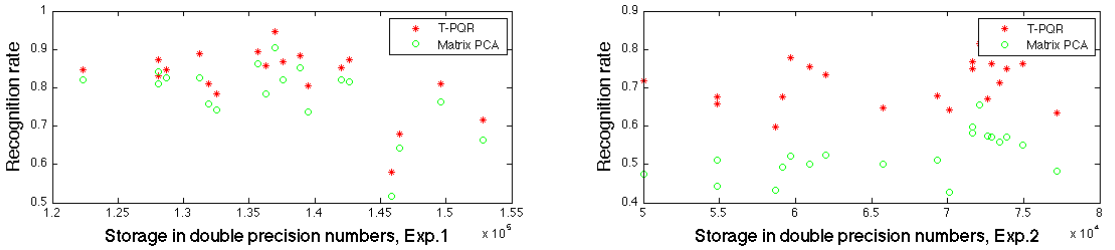

FIG. 7.3. *Recognition Rate and Storage (T-PQR vs PCA), Experiment 1 (left) and Experiment 2 (right)*

are not drastically different, we expect the Fourier coefficients of rows and columns to obey the same type of distribution, and hence we expect that the compression and recognition performance will not deteriorate as a result[3]. To underscore this, using the same testing and training images that resulted in the data in Table 7.1, we repeated the experiment using the transposed images and compared matrix PCA to our T-SVD-based approach in Tables 7.9 - 7.12. Comparing the results in Table 7.1 with those in Table 7.9, we see the compression factors and recognition rates are nearly identical whether we use the tensor with images transposed or not. Comparisons between Table 7.5 and 7.10, Table 7.2 and 7.11 and Table 7.6 and 7.12 reveal the same pattern.

In experiment 1, the first 10 eigenfaces (i.e. columns of $\mathbf{U}_r$ reshaped into images) using PCA

[3]The same could not be said if the images were highly structured - for example, if the images were pictures of, say, vertically oriented layered media, we would expect very different behavior if the images are first transposed.

	Rec Rate	K by T-SVD	Storage for T-SVD	k by PCA	Storage for PCA
mean	0.796	92	57467	38	157852
median	0.829	93	58218	38	155775
max	0.921	101	63226	53	220162
min	0.516	83	51958	31	128774

TABLE 7.9

Comparison between T-SVD (Algorithm 5) and PCA (Algorithm 1) in Experiment 1 with transposed images.

	Rec Rate	K by T-PQR	Storage for T-PQR	k by PCA	Storage for PCA
mean	0.816	216	135341	44	180699
median	0.842	212	132399	45	184853
max	0.947	242	151492	54	224316
min	0.558	196	122696	34	141236

TABLE 7.10

Comparison between T-PQR (Algorithm 6 with compression) and PCA (Algorithm 1) in Experiment 1 with transposed images.

	Rec Rate	K by T-SVD	Storage for T-SVD	k by PCA	Storage for PCA
mean	0.657	97	23727	36	133977
median	0.682	99	24231	35	132090
max	0.758	119	29274	41	154734
min	0.561	73	17958	29	109446

TABLE 7.11

Comparison between T-SVD (Algorithm 5) and PCA (Algorithm 1) in Experiment 2 with transposed images.

	Rec Rate	K by T-PQR	Storage for T-PQR	k by PCA	Storage for PCA
mean	0.693	242	59507	44	167754
median	0.71	240	59040	45	167943
max	0.797	294	72324	54	203796
min	0.581	187	46002	35	132090

TABLE 7.12

Comparison between T-PQR (Algorithm 6 with compression) and PCA(Algorithm 1) in Experiment 2 with transposed images.

are given in Fig. 7.4. Unlike PCA, note that the first 10 lateral slices of $\mathcal{U}$ using the T-SVD do not resemble human faces (see Fig. 7.5). This should not be too surprising given that the reconstruction is given in terms of t-linear, rather than linear, combinations (refer also to Theorem 5.2).

FIG. 7.4. *First 10 PCA eigenfaces , Extended Yale B dataset, Experiment 1*

One possible explanation of the success of our method is that an image is essentially a second

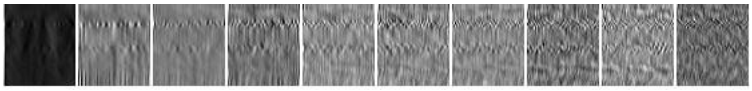

FIG. 7.5. *First 10 lateral slices of $\mathcal{U}$, Extended Yale B dataset, Experiment 1*

order tensor, or matrix, and it is reasonable to consider that pixels close to each other are correlated to some extent. So when we store the entire dataset into a 3rd order tensor and each image as a twisted matrix, we retain the correlation between each pair of pixels.

7.2. Compression Comparison of Truncated Tensor Methods on the Extended Yale B Database. In this example, we will use $Ill = 10$ different illumination conditions of all $P = 38$ people in the Extended Yale B database as whole. Our goal in this section is to try to compare the compressed representations of the image data as given by our proposed T-SVD approach vs. that given by a compressed version of TensorFaces. As mentioned previously, the original TensorFaces algorithm required the full HOSVD of the tensor. To find a compressed representation of the original tensor, we have to decide on a method to truncate the information, and we mentioned previously that there are many different options for doing this. Since we want to try to give as close a comparison as possible with our method, we will use only a 3-way version of TensorFaces here. We will compress the illumination conditions by TensorFaces using truncated mode matrices and compare with our T-SVD method.

For the TensorFaces variant, we store the whole dataset as a third order tensor $\mathcal{F}$ with people on the first mode, illumination on the second mode and pixel on the third mode, and we decompose $\mathcal{F}$ as

$$\mathcal{F} = \mathcal{C} \times_1 \mathbf{U}_p \times_2 \mathbf{U}_i \times_3 \mathbf{U}_{pix}.$$

Note that this is an exact decomposition computed by a direct factorization algorithm, hence the equality. For our T-SVD method, we also store the whole dataset as a third order tensor $\mathcal{T}$ with each lateral slice as an image, and we decompose $\mathcal{T}$ as

$$\mathcal{T} = \mathcal{U} * \mathcal{S} * \mathcal{V}^T.$$

This decomposition is also exact.

Next, we determine a compressed (i.e. approximate) representation for $\mathcal{F}$ by compressing on the second dimension. The second dimension in this case refers to illumination. For a given reduced illumination dimension value K, we choose $\hat{\mathbf{U}}_i = \mathbf{U}_i(:, 1:K)$ and calculate $\hat{\mathcal{C}} := \mathcal{F} \times_1 \mathbf{U}_p^T \times_2 \hat{\mathbf{U}}_i^T \times_3 \mathbf{U}_{pix}$, so that the approximation to the original tensor is

$$\mathcal{F} \approx \hat{\mathcal{C}} \times_1 \mathbf{U}_p \times_2 \hat{\mathbf{U}}_i \times_3 \mathbf{U}_{pix}.$$

In our T-SVD method, the second mode is the number of all images ($P \times Ill$) so we choose the corresponding factor tensors as follows: $\hat{\mathcal{U}} = \mathcal{U}(:, 1:P \times K, :)$, $\hat{\mathcal{S}} = \mathcal{S}(1:P \times K, 1:P \times K, :)$, $\hat{\mathcal{V}} = \mathcal{V}(:, 1:P \times K, :)$ and we have the approximation as

$$\mathcal{T} \approx \hat{\mathcal{U}} * \hat{\mathcal{S}} * \hat{\mathcal{V}}^T.$$

Let each image have size $s_1 \times s_2$, then the storage for TensorFaces and T-SVD are:

$$\texttt{Storage}_{\texttt{TensorFaces}} = \texttt{P} \times \texttt{K} \times (\texttt{P} \times \texttt{Ill}) + \texttt{P} \times \texttt{P} + \texttt{Ill} \times \texttt{K} + (\texttt{s}_1 \times \texttt{s}_2) \times (\texttt{P} \times \texttt{Ill}),$$

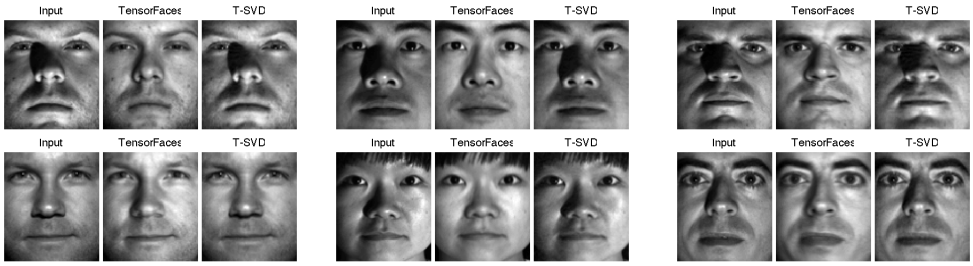

FIG. 7.6. *Comparison of some of the individual images in the original data set with their corresponding compressed representations.*

$$\texttt{Storage}_{\texttt{T-SVD}} = \texttt{s}_1 \times (\texttt{P} \times \texttt{K}) \times \texttt{s}_2 + (\texttt{P} \times \texttt{K})^2 \times \texttt{s}_2 + (\texttt{P} \times \texttt{Ill}) \times (\texttt{P} \times \texttt{K}) \times \texttt{s}_2.$$

In this example, $s_1 = 96$, $s_2 = 84$, $P = 38$, $Ill = 10$, $K = 1$, $\texttt{Storage}_{\texttt{TensorFaces}} = 3080214$, $\texttt{Storage}_{\texttt{T-SVD}} = 1640688$.

In Fig. 7.6, we compare several samples from the compressed data representations with their corresponding original images. Note that our T-SVD method retained more important illumination information without degrading the appearance.

We note again that there are other choices that could have been made to determine a compressed version of TensorFaces. However, as noted previously, each option brings with it a new set of concerns, such as a possible increase in computational complexity for computing the approximate factorization if we desire an optimal reduced dimensional Tucker representation and/or the need to decide on truncation strategies a priori.

7.3. Facial Recognition on Weizmann database. As in the previous section, in order to make a more fair comparison between our tensor method (which is based on a third order tensor representation) and TensorFaces, we implement TensorFaces on the same database formed as a third order tensor. The tensor has three modes specifying people, all other conditions, and pixels, respectively. The total number of other conditions is $C = V \times I \times E$. In this case the core and basis tensors become

$$\mathcal{Z} = \mathcal{L} \times_1 \mathbf{U}_p^T \times_2 \mathbf{U}_c^T \times_3 \mathbf{U}_{pix}^T \tag{7.1}$$

$$\mathcal{B} = \mathcal{Z} \times_2 \mathbf{U}_c \times_3 \mathbf{U}_{pix} \tag{7.2}$$

and the rest follows as above. For supplement, we will also show the result of TensorFaces method by forming the database as a fifth order tensor, using Algorithm 2. In this experiment, we will illustrate our method based on Algorithm 4, but Algorithm 5 can be applied too in a similar way as in the above experiments.

Our first pass at this is to use the recovery ratio [10] to decide a proper truncation index[4]. in tensor based method, say t^*.

$$\rho(\mathcal{A}, \mathcal{U}_{t^*}) = \frac{\sum_{i=1}^{t^*} ||s_i||_F^2}{||\mathcal{A}||_{\mathrm{F}}^2} \tag{7.3}$$

[4]Note the similarity with Eq 5.7. These energy definitions are closely related in that $n \times ||A||_F^2 = ||\hat{A}||_F^2$ for $A \in \mathbb{R}^{\ell \times m \times n}$. For the case where we only do the truncation on t^*, this one is sufficient to use/understand.

FIG. 7.7. *Sample images from Weizmann database*

s_i is the i-th singular tuple of tensor $\mathcal{A}$.

Since the Frobenius norm is invariant under t-products with orthogonal tensors [13]: $||\mathcal{A}||_{\mathrm{F}}^2 = ||\mathcal{U} * \mathcal{S} * \mathcal{V}||_{\mathrm{F}}^2 = ||\mathcal{S}||_{\mathrm{F}}^2 = \sum_i ||s_i||_F^2$, the recovery ratio is the proportion of the energy the first t^* principal components covered over the whole energy.

We test our algorithms on a portion of the Weizmann face database which included 28 different people photographed in 5 viewpoints, 3 illuminations and 3 expressions. We decimated the images by a factor of 3 to avoid an out-of-memory problem using TensorFaces method. A sample of the dataset is shown in Fig. 7.7.

The summary of experiments on the Weizmann database is as follows:

- Experiment 1: Training: 10 people, all 5 viewpoints, the first 2 illuminations, 3 expressions. Testing: 10 people, all 5 viewpoints, the 3rd illuminations, 3 expressions
- Experiment 2: Training: 10 people, 5 viewpoints, 3 illuminations, the first 2 expressions. Testing: 10 people, 5 viewpoints, 3 illuminations, the 3rd expression

We use $t^* = 20$ in T-SVD and T-PQR methods since it recover more than 90% energy based on Equation(7.3) and we did no truncation in TensorFace method. The recognition rate comparison for our method using $t^* = 20$ vs. full TensorFaces described in the beginning of the section is given in Table 7.13.

Experiment	TensorFaces(5th)	TensorFaces(3rd)	T-SVD	T-PQR
Experiment 1	0.8133	0.8033	0.8933	0.84
Experiment 2	0.8933	0.8633	0.9633	0.9633

TABLE 7.13

Recognition rate for $t^* = 20$ for T-SVD (Algorithm 4), TPQR (Algorithm 6) methods, Weizmann dataset.

We note that our tensor method has several potential advantages over the TensorFaces method [32]: we keep the images themselves in multidimensional format, we form only 3rd order tensors but still recover qualitatively similar representations as TensorFaces for the same level of compression, and our T-PQR based approximation to the T-SVD method is specifically easier to update/downdate than the TensorFace method.

The Storage required for full TensorFaces and T-SVD/T-PQR depend on the size of the problem, but we could always turn to Algorithm 5 to do further compression.

8. Conclusions and Future work. In this paper, we briefly reviewed the new tensor-tensor multiplication and several related concepts, such as orthogonal tensors and the T-SVD from [13], and we introduced a new pivoted tensor QR factorization in the hope of identifying a tensor decom-

position with similar compression properties that had superior updating and downdating capabilities. Clearly, other tensor versions of matrix rank-revealing factorizations (e.g. ULV) are possible options as well. We developed a PCA-like analysis of 3rd order tensors using these tensor-tensor decompositions and a framework that is reminiscent of the matrix version. Our examples in facial recognition show that our tensor-based approach gives more information recovered for the same amount of storage as compared with traditional PCA. Even compared to TensorFaces, a recent method for using higher order decompositions for image databases containing different viewpoints, illumination, expressions, we obtained comparable or better recognition results with compressed versions of our third order algorithms.

Since the proposed tensor-tensor methods were PCA-like, we compared our algorithms against traditional matrix PCA and the closest tensor-based PCA analogue, TensorFaces. We note, however, that even the matrix-based PCA approach can be augmented to improve performance; for example, recent work by Wright *et al.* [34] has shown remarkable improvement in a matrix-based approach when sparsity constraints are added to the mix, at the expensive of having to solve an optimization problem to extract the features. In future work, we will consider adding analogous constraints to our tensor-based framework and plan to make direct comparison with these sophisticated, matrix-based algorithms.

Recent work by Martin [22] has shown that the tensor-based factorizations here can be defined in a recursive manner for higher-order tensors. This suggests the possibility of adding more dimensions to our database, similar to what is done with TensorFaces. Future work will therefore focus on extending the approach presented here to higher order tensors and comparison with other tensor-based algorithms such as the method in [24].

9. Acknowledgements. The authors would like to thank the two anonymous reviewers and the Associate Editor whose comments helped us to greatly improve the quality of this manuscript.

REFERENCES

[1] *The Extended Yale Face Database B.* http://vision.ucsd.edu/ ˜leekc/ExtYaleDatabase/ExtYaleB.html.
[2] Karen Braman, *Third-order tensors as linear operators on a space of matrices*, Linear Algebra and its Applications, (2010), pp. 1241–1253.
[3] J.D. Carroll and J. Chang, *Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition*, Psychometrika, 35 (1970), pp. 283–319.
[4] J. Chen and Y. Saad, *On the tensor SVD and the optimal low rank orthogonal approximation of tensors*, SIAM Journal on Matrix Analysis and Applications, 30 (2009), pp. 1709–1734.
[5] P. Comon, *Tensor decompositions*, in Mathematics in Signal Processing V, J. G. McWhirter and I. K. Proudler, eds., Clarendon Press, Oxford, UK, 2002, pp. 1–24.
[6] A.S. Georghiades, P.N. Belhumeur, and D.J. Kriegman, *From few to many: Illumination cone models for face recognition under variable lighting and pose*, IEEE Trans. Pattern Anal. Mach. Intelligence, 23 (2001), pp. 643–660.
[7] Gene Golub and Charles Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, 3rd edition ed., 1996.
[8] R.A. Harshman, *Foundations of the parafac procedure: Model and conditions for an 'explanatory' multi-mode factor analysis*, UCLA Working Papers in phonetics, 16 (1970), pp. 1–84.
[9] W. Scott Hoge and Carl-Fredrik Westin, *Identification of translational displacements between N-dimensional data sets using the high order SVD and phase correlation*, IEEE Transactions on Image Processing, 14 (2005), pp. 884–889.
[10] Randy Hoover, Karen Braman, and Ning Hao, *Pose estimation from a single image using tensor decomposition and an algebra of circulants*, Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, (2011).

[11] Misha E Kilmer, Karen Braman, and Ning Hao, *Third order tensors as operators on matrices: A theoretical and computational framework with applications in imaging*, Tufts Computer Science Technical Report, (2011).

[12] Misha E. Kilmer, Karen Braman, Ning Hao, and Randy C. Hoover, *Third order tensors as operators on matrices: A theoretical and computational framework with applications in imaging*. In review with SIAM Journal on Matrix Analysis and Applications, 2011.

[13] Misha E. Kilmer and Carla D. Martin, *Factorization strategies for third-order tensors*, Linear Algebra and its Applications, Special Issue in Honor of G. W. Stewart's 70th birthday, Vol. 435 (2011), pp. 641–658. DOI: 10.1016/j.laa.2010.09.020.

[14] Misha E Kilmer, Carla D. Martin, and Lisa Perrone, *A third-order generalization of the matrix svd as a product of third-order tensors*, Tufts Computer Science Technical Report, (2008).

[15] T.G. Kolda and B.W. Bader, *Higher-order web link analysis using multilinear algebra*, Proceedings of the 5th IEEE International Conference on Data Mining, ICDM 2005,IEEE Computer Society (2005), pp. 242–249.

[16] ———, *Tensor decompositions and applications*, SIAM Review, 51 (2009), pp. 455–500.

[17] P.M. Kroonenberg, *Three-mode principal component analysis: Theory and applications*, DSWO Press, Leiden, 1983.

[18] L. De Lathauwer, B. De Moor, , and J. Vandewalle, *A multilinear singular value decomposition*, SIAM Journal of Matrix Analysis and Applications, 21 (2000), pp. 1253–1278.

[19] L. De Lathauwer and B. De Moor, *From matrix to tensor: Multilinear algebra and signal processing*, in Mathematics in Signal Processing IV, J. McWhirter and eds. I. Proudler, eds., Clarendon Press, Oxford, UK, 1998, pp. 1–15.

[20] L. De Lathauwer, B. De Moor, and J. Vandewalle, *On the best rank-*1 *and rank-*$R_1, \ldots, R_N$ *approximation of higher-order tensors*, SIAM J. Matrix Anal. Appl., 21 (2000), pp. 1324–1342.

[21] K.C. Lee, J. Ho, and D. Kriegman, *Acquiring linear subspaces for face recognition under variable lighting*, IEEE Trans. Pattern Anal. Mach. Intelligence, 27 (2005), pp. 684–698.

[22] Carla D. Martin, Richard Shafer, and Betsy LaRue, *A recursive idea for multiplying order-p tensors*, SIAM Journal on Scientific Computing (submitted July 2011).

[23] J.G. Nagy and M.E. Kilmer, *Kronecker product approximation for preconditioning in three-dimensional imaging applications*, IEEE Trans. Image Proc., 15 (2006), pp. 604–613.

[24] Xiaofei He Deng Cai Partha Niyogi, *Tensor subspace analysis*, Advances in Neural Information Processing Systems, 18 (2006), pp. 499–506.

[25] Karl Pearson, *On lines and planes of cloest fit to systems of points in space*, Philosophical Magazine, 2 (1901), pp. 559–572.

[26] M. Rezghi and L. Eldèn, *Diagonalization of tensors with circulant structure*, Linear Algebra and its Applications, Special Issue in Honor of G. W. Stewart's 70th birthday (2010). In Press. Available On-line Oct. 2010.

[27] N. Sidiropoulos, R. Bro, and G. Giannakis, *Parallel factor analysis in sensor array processing*, IEEE Transactions on Signal Processing, 48 (2000), pp. 2377–2388.

[28] A. Smilde, R. Bro, and P. Geladi, *Multi-way Analysis: Applications in the Chemical Sciences*, Wiley, 2004.

[29] L.R. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311.

[30] M. Turk and A. Pentland, *Face recognition using eigenfaces*, in Proc. IEEE Conference on Computer Vision and Pattern Recognition, 1991.

[31] M.A.O. Vasilescu and D. Terzopoulos, *Multilinear analysis of image ensembles: Tensorfaces*, Proceedings of the 7th European Conference on Computer Vision ECCV 2002, (2002), pp. 447–460. Vol. 2350 of Lecture Notes in Computer Science.

[32] ———, *Multilinear image analysis for face recognition*, Proceedings of the International Conference on Pattern Recognition ICPR 2002, 2 (2002), pp. 511–514. Quebec City, Canada.

[33] ———, *Multilinear subspace analysis of image ensembles*, Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR 2003, (2003), pp. 93–99.

[34] J. Wright, A. Yang, A. Ganesh, S. Sastry, and Y. Ma, *Robust face recognition via sparse representation*, IEEE Trans. Pattern Analysis and Machine Intelligence, 31 (2009), pp. 210–227.